State-Based Disassembly Planning

Chao Lei, Nir Lipovetzky, Krista A. Ehinger

School of Computing and Information Systems, The University of Melbourne, Australia
clei1@student.unimelb.edu.au, {nir.lipovetzky, kris.ehinger}@unimelb.edu.au

Abstract

It has been shown recently that physics-based simulation significantly enhances the disassembly capabilities of real-world assemblies with diverse 3D shapes and stringent motion constraints. However, the efficiency suffers when tackling intricate disassembly tasks that require numerous simulations and increased simulation time. In this work, we propose a State-Based Disassembly Planning (SBDP) approach, prioritizing physics-based simulation with translational motion over rotational motion to facilitate autonomy, reducing dependency on human input, while storing intermediate motion states to improve search scalability. We introduce two novel evaluation functions derived from new *Directional Blocking Graphs* (DBGs) enriched with state information to scale up the search. Our experiments show that SBDP with new evaluation functions and DBGs constraints outperforms the state-of-the-art in disassembly planning in terms of success rate and computational efficiency over benchmark datasets consisting of thousands of physically valid industrial assemblies.

Introduction

Assembly planning is crucial for optimizing manufacturing efficiency and minimizing maintenance and repair costs (Ghandi and Masehian 2015). Numerous methods for automated assembly have been proposed (Perrard et al. 2023; Bedeoui et al. 2018; Tian et al. 2022) through various methodologies, such as heuristic search (Wang et al. 2013), evolutionary algorithms (Zeng et al. 2011) and neural networks (Chen et al. 2008). Despite these efforts, automatically generating an efficient, precise and generalizable assembly plan remains an open challenge. An assembly planning task often contains two steps: assembly sequence planning, which computes the sequential order to assemble all components; and assembly path planning, which searches for penetration-free trajectories to add new components into a sub-assembly. The bijection between assembly and disassembly sequences, when all parts are rigid, raises the idea of assembly-by-disassembly, introduced by Homem de Mello and Sanderson (1991). The assembly-by-disassembly method, understood as regression, minimizes the required search space in assembly tasks by leveraging the defined precedence and motion constraints in assembled parts. It

has been widely implemented in different assembly tasks, including mechanical product assembly (Homem de Mello and Sanderson 1991; Wang, Rong, and Xiang 2014) and kit assembly (Zakka et al. 2020).

With the development of general-purpose assembly planning algorithms, Tian et al. (2022) proposed a physics-based assembly-by-disassembly planning method, referred as PDP, for *sequential* assembly tasks where operations involve inserting an individual part into the sub-assembly. This approach employs a physics-based simulation, built upon the rigid body simulator developed by Xu et al. (2021) to generate the disassembly trajectories while using a sequential planner to determine the disassembly sequence. The disassembly trajectories and sequence are reversed to construct the assembly plan. Compared to sampling-based approaches such as RRT (LaValle 1998) and its variants (Aguinaga, Borro, and Matey 2008; Ebinger et al. 2018), as well as the classic physics-based method (Zickler and Veloso 2009), PDP demonstrates state-of-the-art performance in terms of success rate and computational efficiency.

The exhaustive search in disassembly sequence planning and path planning results in redundant and unnecessary simulations in PDP. Regenerating already simulated trajectories instead of reusing them also weakens the efficiency of PDP. In addition, users must specify if a task requires translational or rotational operations, as considering both at the same time hinders the advantages of using one operation at a time, where translational motion offers computational efficiency, while rotational motion provides better coverage. Requiring user input reduces the autonomy of the deployed solutions in PDP. To address these issues, we prioritize translational motion over rotational motion in a State-Based Disassembly Planning search algorithm. We use the physical simulation as a transition function and record the trajectories in the state representation, avoiding computationally intensive simulations of states that need to be revisited. We integrate and propose novel *Directional Blocking Graphs* with state information to derive new evaluation functions that allow disassembly planning to accurately select components and actions for disassembly.

Background

A common strategy for solving planning problems is to create a state model for the target domain (Bonet and

Geffner 2001). Formally, a state model is a tuple $\mathcal{S} = \langle S, s_0, S_G, A, f, c \rangle$ where $\mathcal{S}$ consists of a set of states S, a set of actions A, where $A(s) \subseteq A$ are applicable actions in each state $s \in S$, an initial state $s_0 \in S$, a set of goal states $S_G \subseteq S$, a deterministic transition function $s' = f(a, s)$ that defines how action $a \in A(s)$ map one state s into another state s', and the cost function $c(a, s)$ that estimates the cost of applying action a in state s. The state model is well-suited for the disassembly planning task given that actions are defined and disassembly configurations are fully observed.

We formulate a *sequential* disassembly problem with a state model. A disassembly problem $\mathcal{P}$ consisting of m parts $P = \{p_1, \ldots, p_m\}$ is described by a tuple $\mathcal{P} = \langle P, S, A, s_0, S_G, f \rangle$, where $S = (\vec{s}_1, \ldots, \vec{s}_m)^T \in \mathbb{R}^{3 \times m}$ describes the translational vector of each part for translational motion problems with actions $A = \{(\pm 1, 0, 0), (0, \pm 1, 0), (0, 0, \pm 1)\}$; otherwise, $S = SE(3)^m$ describes the transformation matrix of each part with actions $A = \{(\pm 1, 0, 0, 0, 0, 0), \ldots, (0, 0, 0, 0, 0, \pm 1)\}$, corresponding to 6 translational and 6 rotational degrees of freedom, for rotational motion problems. Each action specifies a positive or negative change in one dimension for a single movable part $p \in P$. s_0 denotes the initial assembled state of all parts $s_0 \in S$. $s' = f(a, s, p)$ is the state transition function implemented by the physical simulation. The physical simulation generates a penetration-free motion trajectory (path) from s to s', denoted as $t(s, s', a, p)$, by continuously moving p from state s with action $a \in A$, and ultimately returning a state s'. We call s' as a collision state if p at s' has a collision with any other components, or a disassembled state that satisfies the geometry constraint, iff the convex hull of the geometry of part p at s' does not intersect with the convex hull encompassing the geometries of all other components $P \backslash p$ at s'. $S_G \subseteq S$ specifies the set of disassembled goal states defined as the states that satisfy the geometry constraint for every part in P. A partial disassembly path $\mathcal{T}_p = \{t(s_0, s_1, a_0, p) \ldots, t(s_{n-1}, s_n, a_{n-1}, p)\}$, consisting of a sequence of motion paths for a single part p, is a disassembly path when s_n is the disassembled state for p. A disassembly plan, $\mathcal{DP} = \{\mathcal{T}_{p_1}, \ldots, \mathcal{T}_{p_m}\}$, comprises a sequence of ordered disassembly paths for every part in P.

We note that once the translational vector or transformation matrix encoded in a state is provided, the specific location can be determined by applying it to the original coordinates. In other words, the state $s = \{s_{p_1}, \ldots, s_{p_m}\}$ describes the location of each part, where we use s_{p_m} to represent the location of p_m at state s. s is the initial state s_0 when the location of each part in s is the initial location in $\mathcal{P}$. s is a goal state s_g when each part at its location in s meets the geometry constraint. The transition function $s' = f(a, s, p_1)$ updates the state $s = \{s_{p_1}, \ldots, s_{p_m}\}$ to the state $s' = \{s'_{p_1}, \ldots, s'_{p_m}\}$ through the physics-based simulation, where only s'_{p_1} is changed while $\{s'_{p_2}, \ldots, s'_{p_m}\} = \{s_{p_2}, \ldots, s_{p_m}\}$ as only p_1 is simulated. Subsequently, only p_1 is available to simulate in state s' since we only consider the sequential disassembly problem.

To keep the simulation computationally manageable, we follow the same assumptions as introduced by Tian et al.

(2022): 1) assemblies consist entirely of rigid components; 2) gravity, friction and manipulation constraints are omitted; 3) parts can be fully assembled or disassembled sequentially.

Physics-Based Disassembly Planning

PDP leverages a breadth-first search (BFS) as the path planner to disassemble one part at a time. In the search tree, each node represents the location configuration of an assembled part, and each edge corresponds to the physical simulation with an action that updates the location of an assembled part, resulting in a child node. Given an assembled part p, the path planner searches for a sequence of actions until a disassembled state has been found or a depth limit on the longest sequence has been reached. If a part p is successfully disassembled, its disassembly path $\mathcal{T}_p$ is attached to the disassembly plan $\mathcal{DP}$, and the planner proceeds to the next part. The sequence planner in PDP iteratively applies the path planner for each assembled part until either all parts have been disassembled or a timeout is reached. To limit the BFS search tree growth over obstructed parts that cannot be disassembled prior to other parts, the sequence planner restricts the maximum depth d_{max} of a BFS tree. The depth limit starts with $d_{max} = 1$, and increments if no disassembled states have been found after searching for a disassembly path over all parts. We note that PDP reconstructs the entire search tree from scratch every time d_{max} is incremented.

Physics-Based Simulation. In PDP, the motion trajectory is generated by the physics-based simulation that confines path planning to explore a physically valid subspace rather than the entire state space as in the sampling-based method (LaValle 1998). Given a starting state s, an action a, and a candidate movement part p, the simulator iteratively applies a over the part p at state s with a specified kinematic time step, producing a series of valid motion locations. The simulation ultimately returns either a reached disassembled state for part p or a collision state when the distance between two generated locations is within the collision threshold.

Collision Detection. To simulate contact-rich disassembly tasks, the physics-based engine uses a Signed Distance Field (SDF) representation for each part. SDF enables accurate collision distance computation during trajectory generation. SDF $g_p(pt) : \mathbb{R}^3 \to \mathbb{R}$ associates a point $pt \in \mathbb{R}^3$ with its closest distance to a part p, where a negative value indicates that pt is inside the geometry of p. SDF improves the collision distance computation for intricate assembly geometries, such as screws with fine threads (Tian et al. 2022).

Directional Blocking Graph

A *Directional Blocking Graph* (DBG), introduced by Wilson (1992), defines a vertex as an individual part and edges as blocking relations given a specific movement direction. In a $\text{DBG}(d)$, an edge $p_i \to p_j$ indicates that the part p_j obstructs the translation of part p_i along the direction d. Consequently, p_i with an outdegree of zero, i.e. a sink vertex, indicates that a collision-free disassembly path exists for p_i in direction d. DBGs can reduce the complexity of the disassembly planning problem as all precedence constraints between components are identified. Different approaches have
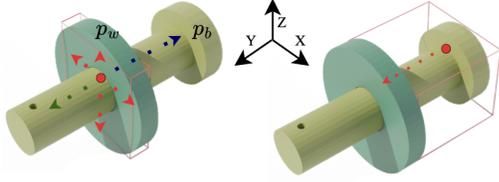
Figure 1: An illustration of DBGs static analysis (left) with respect to a bolt p_b and a washer p_w along six directions; a demonstration of potential blocking assessment (right).

Algorithm 1: SBDP

Input: A disassembly problem $\mathcal{P}$ consisting of m parts P with the initial assembled state s_0, timeout $t_{\max}$, translational actions A_t, rotational actions A_r.

Output: An ordered sequence of disassembly paths $\mathcal{DP} := \{\mathcal{T}_{p_1}, \ldots, \mathcal{T}_{p_m}\}$.

1   $\mathcal{DP} := \mathcal{T} := \Pi_t := \Theta_t := \Pi_r := \Theta_r := \emptyset; P_r := P;$
2   $\Pi_t := \Pi_t \cup (s_0, P_r, \mathcal{T}); \Pi_r := \Pi_r \cup (s_0, P_r, \mathcal{T});$
3   **if** s_o is the goal state **then**
4   **return** $\mathcal{DP}$
5   **Function** $disassembly\,(\Pi, \Theta, A)$ **:**
6   **while** $\Pi \neq \emptyset$ **do**
7        $s, P^*, \mathcal{T} := extractNode(\Pi);$
8        $\Theta := \Theta \cup (s, P^*, \mathcal{T});$
9   **if** time $t > t_{\max}$ **then**
10   **return** $\Pi, \Theta, \text{true}, \emptyset$
11   **for** part p in P^* **do**
12   **for** action a in A **do**
13            $s', t(s, s', a, p) := f(a, s, p);$ // Simulator
14            $\mathcal{T}_p := \mathcal{T} \cup t(s, s', a, p);$
15   **if** $isDisassembledState(s', p, P_r)$ **then**
16              $\mathcal{DP} := \mathcal{DP} \cup \mathcal{T}_p; P_r := P_r \setminus p;$
               // Delete nodes where $P^* = \{p\}$
17              $Delete(p);$
18   **if** $isGoalState(P_r)$ **then**
19   **return** $\Pi, \Theta, \text{false}, \mathcal{DP}$
20              $\Pi := \Pi \cup \Theta; rmd(\Pi); \Theta := \emptyset;$
               // Terminate for Rotation
21   **if** $isRotational$ **then**
22   **return** $\Pi, \Theta, \text{false}, \emptyset$
23   **else**
               // The collision state
24              $\Theta := \Theta \cup (s', \{p\}, \mathcal{T}_p)$
25      $\Pi := \Pi \cup \Theta; rmd(\Pi); \Theta := \emptyset;$
26   **return** $\Pi, \Theta, \text{false}, \emptyset$
27 **while** true **do**
     // Translational disassembly planning
28      $\Pi_t, \Theta_t, \text{timeout}, \text{goal} := disassembly\,(\Pi_t, \Theta_t, A_t);$
29   **if** timeout **then return** failed;
30   **if** goal $\neq \emptyset$ **then return** $\mathcal{DP}$;
     // Rotational disassembly planning
31      $\Pi_r, \Theta_r, \text{timeout}, \text{goal} := disassembly\,(\Pi_r, \Theta_r, A_r);$
32   **if** timeout **then return** failed;
33   **if** goal $\neq \emptyset$ **then return** $\mathcal{DP}$;

---

been introduced to build DBGs based on geometric and spatial analysis such as Minkowski differences (Lozano-Perez and Wilson 1993; Wilson et al. 1995).

State-Based Disassembly Planning

State-Based Disassembly Planning (SBDP) approach stores the unexplored states in an *open list* Π and the resulting states, including visited states and simulated collision states, in a *collision list* Θ. SBDP explores disassembly paths of varying lengths by expanding stored states rather than a FIFO policy constrained with d_{max} that needs to reconstruct the entire search tree every time d_{max} is updated. SBDP expands one node at a time. When the initial state is expanded, it tries to disassemble all parts. Otherwise, only part p is enabled for disassembly when the expanded state results from moving p. SBDP integrates the efficiency of translational motion and the coverage of rotational motion, by prioritizing disassembly planning with translational motion, denoted as translational disassembly planning, over disassembly planning with rotational motion, named as rotational disassembly planning. SBDP leverages translational disassembly planning to repeatedly disassemble each part until no further disassembly is feasible, and then applies rotational disassembly planning to disassemble remaining assemblies. To keep efficiency in SBDP, rotational disassembly planning terminates upon the removal of one part, followed by translational disassembly planning to disassemble remaining assemblies. In SBDP, translational planning and rotational planning maintain the same disassembly plan $\mathcal{DP}$ and remaining assemblies, ensuring consistency in the parts requiring disassembly in either planning, and allowing $\mathcal{DP}$ to contain a mix of translational and rotational motions, through the concatenation of individual plans.

Algorithm 1 shows the pseudo-code of SBDP. The main loop, Lines 27-33, repeatedly executes translational disassembly planning (Line 28) and rotational disassembly planning (Line 31) with the rule that rotational planning is considered only after completing translational planning. SBDP implements both translational and rotational planning within the disassembly function (Lines 5-26). Line 1 initializes the open list Π, collision list Θ, disassembly plan $\mathcal{DP}$ and partial disassembly path $\mathcal{T}$ with the empty set. The open list Π is updated with an unexplored node tuple $\Pi = \{(s_0, P_r, \mathcal{T})\}$ (Line 2), where P_r denotes remaining assemblies, initialized with P. We use subscripts t and r to denote Π, Θ and actions A used in translational and rotational planning. SBDP returns an empty $\mathcal{DP}$ if the initial state is a goal state (Lines 3-4), or a disassembly plan $\mathcal{DP}$ on Line 30 or 33 when translational or rotational planning achieves the goal state in the disassembly function (Lines 18-19); otherwise, a $failed$ is returned on Line 29 or 32 after reaching timeout in the disassembly function (Lines 9-10) for either planning approach.

The disassembly function receives the open list Π, collision list Θ, and disassembly actions A from either planning approach as input and returns updated Π and Θ on Line 26 after moving newly generated nodes from Θ to Π, removing duplicate nodes in Π ($rmd(\Pi)$) and clearing Θ on Line 25. Lines 5-26 detail the implementation of the disassembly function. An unexplored node, containing a state s, candidate disassembly parts P^*, and a partial disassembly path $\mathcal{T}$, is selected and removed from the open list Π on Line 7, where $P^* = P_r = P$ when $s = s_0$. The expanded node is inserted into the collision list Θ on Line 8 to record the vis-
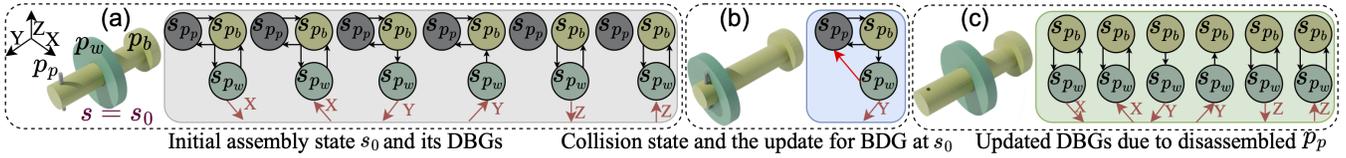
Figure 2: DBGs examples with respect to construction through static analysis (grey) and updates due to the collision state (blue) and the disassembled state (green) in a problem $\mathcal{P}$ that consists of a bolt p_b, a washer p_w, and a pin p_p.

ited state. SBDP tries to disassemble a part $p \in P^*$ with an action $a \in A$, through the transition function $f(a, s, p)$ on Line 13. The transition function updates state s to s' implemented by the physics-based simulation, and the simulated motion path t from s to s' is added to the partial disassembly path $\mathcal{T}$ (Line 14). If s' is a disassembled state for p, SBDP appends the disassembly path $\mathcal{T}_p$ to $\mathcal{DP}$, updates remaining assemblies P_r with $P_r \setminus p$ (Line 16), and deletes nodes where $P^* = \{p\}$ from the open and collision lists of both planning approaches ($\Pi_t, \Pi_r, \Theta_t, \Theta_r$) (Line 17), as there is no need to find other disassembly paths for part p. For the same reason, SBDP updates P^* with $P^* \setminus p$ for the node where $s = s_0$ in the *Delete* function (Line 17). If s' is not the goal state, Line 20 transfers nodes from Θ to Π, eliminates duplicates in Π, and resets Θ to an empty set. This enables the continued search to reassess disassembly across all states in **translational planning**, since a disassembled part may unblock disassembly paths for other parts. We note that for **rotational planning**, SBDP terminates the disassembly (Lines 21-22) upon successfully disassembling one component to restart translational planning and reevaluate potential released blockages, as rotational planning is computationally demanding. If s' is a collision state, SBDP inserts the node tuple $(s', \{p\}, \mathcal{T}_p)$ into Θ (Line 24), resulting in only part p being considered for disassembly when s' is expanded again. We note that for all other nodes where $s \neq s_0$, P^* is set to $\{p\}$, with $p \in P$, as once a failed disassembly of part p leads to a new node, that node and all its child nodes commit to finding a disassembly path for p. Disassembly function terminates, returns updated Π and Θ, and triggers the switch between translational and rotational planning when no part can be disassembled after searching for a disassembly path over all states in Π, leading to $\Pi = \emptyset$.

Compared with PDP, SBDP stores collision states to facilitate the disassembly path generation from the latest reached states rather than reconstructing the entire search tree from the initial state, which scales up the search significantly. To obtain a high-quality disassembly path, i.e. a short path, while minimizing simulation computational cost, SBDP records visited states but removes duplicate states. This allows SBDP to disassemble each part, starting from its initial state and all intermediate motion states. SBDP achieves the goal state when remaining assemblies P_r contain only two parts, and one of them is disassembled through either translational or rotational planning.

DBGs in State-Based Disassembly Planning

Different from previous work where DBGs are precomputed prior to the search using geometric and spatial relations analysis, we introduce a method to build a DBG for each action (moving direction) based on the SDF representation, and a function to update DBGs using feedback from each simulation. In addition, we enrich DBGs in SBDP with state information, where SBDP constructs and updates DBGs for each state to capture the blockage relationships among different parts. For example, in a DBG(a) associated with state s, its vertex is represented by s_p, the location of a part p at s, associated with a potential blocking relationship via moving p at s along the direction defined by the planning action a. We introduce two novel evaluation functions derived from DBGs to guide SBDP. New evaluation functions enable SBDP to extract best-evaluated nodes from the open list in both translational and rotational planning on Line 7 in Algorithm 1, and avoid unnecessary simulations when collisions are detected and recorded in DBGs.

DBGs Static Analysis

SBDP performs static analysis to build DBGs for each state. In static analysis, we consider all pairs of *contacting* parts (p_o, p_s) as potential blocking parts when an overlap exists between their collision shapes defined by axis-aligned minimum bounding boxes given their locations (s_{p_o}, s_{p_s}) at state s. To establish the blocking relations between p_o and p_s, one part, p_o, is designated as mobile, while the other, p_s, is considered stationary. We denote the mesh vertices of part p_o within the overlapping area as contacting points C_{p_o} and extend each contacting point $c_{p_o} \in C_{p_o}$ in disassembly directions over a specified number of steps resulting in a series of extension points E_{p_o}. If there is an extension point $e_{p_o} \in E_{p_o}$ whose SDF distance $g_{p_s}(e_{p_o}) < 0$, then a blocking relationship is built with an edge $s_{p_o} \rightarrow s_{p_s}$ added in the DBG of the extended direction and an edge $s_{p_s} \rightarrow s_{p_o}$ added in the DBG of the opposite direction. For example in the left part of Figure 1, the red lines delineate the overlapping collision shape between the mobile bolt p_b and the stationary washer p_w. A contacting point in p_b and its six extension directions are shown, where the red arrow indicates a detected collision with p_w, and the blue arrow denotes the correct disassembly direction for p_b.

In certain situations, detecting blocking relations based solely on contacting points may be insufficient. For example, the bolt in Figure 1 is blocked along the positive Y direction, indicated with green, due to the bolt head. To address such scenarios, a further blocking assessment is considered, if the collision shape of mobile part p_o is larger than the collision shape of stationary part p_s along the moving axis. We first identify the potential collision area where the collision shape of p_o lies behind the collision shape of p_s in the movement

direction. We then denote the mesh vertices of p_o within the potential collision area as potential contacting points and repeat the same extension process as mentioned above to detect collisions. For example, in the right part of Figure 1, we illustrate a potential contacting point in the red-outlined potential collision area, along its extension direction resulting in a blocking relationship.

In SBDP, static analysis is computed for the initial state and again for each generated collision state. To keep static analysis scalable, we leverage an adaptive extension distance in each extension step, which is a function of the length of the mobile part along the moving axis. Figure 2 (a) displays a problem $\mathcal{P}$ that consists of a bolt p_b, a washer p_w, and a pin p_p, and its six DBGs generated through static analysis at s_0 with respect to six translational actions. Static analysis allows SBDP to construct new DBGs for each state and further avoid the simulations when collisions have already been detected and recorded in the constructed DBGs. However, static analysis may overlook certain collisions due to the constraints on contacting pairs and the number of extension steps for the (potential) contacting point. This will be addressed and complemented in the DBGs updates function.

DBGs Updates

DBGs undergo updates during motion planning according to the result of the transition function computed by physical simulation. If the transition function $f(s, a, p)$ leads to a collision state by moving part p at state s with action a, SBDP updates DBG(a) at s by adding edges from s_p to the colliding parts. Figure 2 (b) illustrates a simulation outcome where the movement of p_w at s_0 along the positive-Y direction ($\nearrow$) leads to a collision between p_p and p_w, and the according DBG update, adding an arc $s_{p_w} \rightarrow s_{p_p}$ in the DBG($\nearrow$) at s_0. In contrast, if $f(s, a, p)$ results in a disassembled state, all DBGs in SBDP are updated, where all vertices and edges related to p are removed, indicating the release of all precedence constraints associated with p. For example, Figure 2 (c) depicts DBGs, resulting from deleting all vertices and edges related to p_p, when p_p is disassembled. We note that DBGs blockage updates are asymmetric since trajectories generated by physical simulation differ for colliding components along the opposite movement direction. For instance in Figure 2 (a), the symmetric simulation over p_p along the negative-Y direction would not lead to the collision with p_w due to the initial obstruction with p_b.

DBGs Heuristics

We introduce two evaluation functions in SBDP given a state s and its DBGs: $f_a(s_p)$ is the number of DBGs where s_p is a sink vertex, i.e. the number of available collision-free actions for part p at the location in state s; $f_c(s_p)$ is the number of parts $p' \neq p$ pointed by the outgoing arcs of s_p over all DBGs associated with state s, i.e. the number of colliding parts with part p at the location in state s. The combination of (f_c, f_a) is used to prioritize the open list, using f_c first, breaking ties with f_a. Specifically, the (f_c, f_a) values for the node $(s, \{p\}, \mathcal{T}_p)$ are computed only for the part p at state s, and the (f_c, f_a) values for the node $(s_0, P^*, \mathcal{T})$ are set to infinite. Besides, (f_c, f_a) is used to order the disassembly
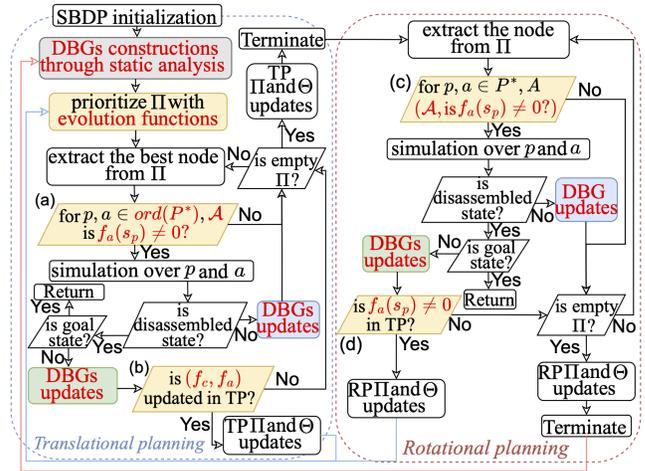

Figure 3: A flow chart of DBG-guided SBDP. DBGs constructions and updates are highlighted with the same colors used in Figure 2 and usages are highlighted with yellow in both translational planning (left) and rotational planning (right). TP and RP are acronyms of translational planning and rotational planning; Π and Θ are open list and collision list. We use parallelogram to denote conditional operation.

sequence on Line 11 in Algorithm 1, resulting in an ordered P^*, when s_0 is the expanded state. f_c and f_a are treated as heuristics, so smaller values are preferred. This encourages SBDP to explore a part with fewer obstructing components and fewer disassembly available actions, as having more disassembly actions would lead to more simulation computation via longer feasible trajectories.

DBGs Constraints

DBGs generated by static analysis and blockage updates enable SBDP to avoid unnecessary physics-based simulations over collision-detected parts with inapplicable actions. For a part p at the expanded state s, SBDP allows the simulation to be computed with an action a if a is a collision-free action for part p, represented by the sink vertex s_p in the DBG(a) associated with state s. This constraint reduces the number of applicable actions A on Line 12 in Algorithm 1, where we use $\mathcal{A}$ to denote satisfied actions. Furthermore, SBDP skips all simulations over part p at the expanded state s if no collision-free actions are available for part p, indicated by $f_a(s_p) = 0$. This constraint prunes the duplicate state produced from the visited state in both translational and rotational planning. For instance, blockage updates for DBGs at an expanded state s record all collisions as a result of disassembly attempts of part p, resulting in $f_a(s_p) = 0$. Consequently, all simulations are skipped due to $f_a(s_p) = 0$ when s is expanded again. We note that the value of $f_a(s_p)$ will be updated through DBGs updates with respect to a disassembled component.

DBG-Guided Disassembly Planning

In SBDP, translational and rotational search states maintain their separate DBGs, but successful disassembly of a part

	All (4196)	Small (2620)	Medium (1469)	Large (107)	Difference with SBDP*
SBDP*	**4135**	**2603**	**1444**	**88**	-0/+0
SBDP	4106	2599	1425	82	-36/+7
PDP*	3976	2551	1348	77	-160/+1
PDP_r	3961	2551	1344	66	-175/+1
PDP_t	3772	2460	1247	65	-363/+0

Table 1: The number of disassembly problems solved by SBDP*, SBDP, PDP*, PDP_r, and PDP_t in the small, medium, and large categories; The last column describes the number of problems solved by SBDP* while remaining unsolved by the planner in each row vs. problems solved by the planner in each row while remaining unsolved by SBDP*.

updates all DBGs. Static analysis is only computed for translational planning, while it is omitted in rotational planning due to the computationally intense nature of rotation analysis. DBGs are set to empty whenever static analysis is required in rotational planning. However, rotational blockages are computed lazily while searching through DBGs updates.

The flowchart in Figure 3 describes DBGs constructions, updates and usages in SBDP, where some SBDP specifics are eliminated to save space. SBDP initiates translational planning with static analysis for the initial state to construct its DBGs. Subsequently, the open list is prioritized based on (f_c, f_a). For each node extracted from the prioritized open list, translational planning performs the simulation when $f_a(s_p) \neq 0$ over the ordered P^* and constrained translational actions $\mathcal{A}$ (block (a)). For each collision state, updated DBGs record the simulated blockages to avoid redundant simulations. If a disassembled state leads to (f_c, f_a) value updates, i.e. a new collision-free action, for states in the open or collision list (block (b)), translational planning continues the search with an updated open list (Line 20 in Algorithm 1) that is prioritized again according to each state's new DBGs (blue flowline). For rotational planning, the absence of static analysis results in an exhaustive search over candidate disassembly parts P^* and rotational actions A (block (c)). However, the updated blockages in DBGs allow rotational planning to focus on constrained actions $\mathcal{A}$ and simulations where $f_a(s_p) \neq 0$ when exploring visited states. Rotational planning reactivates translational planning to reorder its open list while skipping static analysis (blue flowline), when a disassembled state results in any collision-free actions, indicated by $f_a(s_p) \neq 0$, for at least one state in the translational planning open list (block (d)). Otherwise, SBDP restarts translational planning to build DBGs for each generated collision state in its open list through static analysis after the completion of rotational planning (red flowline).

DBGs enable an informed SBDP with a reduced search space. The recorded blockage relationships in DBGs inform SBDP to reassess disassembly in translational planning only for states whose DBGs contain collision-free actions, whether existing or newly generated by the removal of a component in either translational or rotational planning.

Evaluation

Tian et al. (2022) introduced a large-scale dataset comprising thousands of physically valid industrial assemblies, with comprehensive geometric pre-processed data for collision detection. We catergorize assemblies consisting of 3-9 (small), 10-49 (medium), and 50+ (large) components as disassembly benchmarks, with 4196 problems in total. We denote PDP_t and PDP_r as PDP with translational or rotational motion, respectively. PDP_t, PDP_r and the variant PDP* serve as baselines, where PDP* leverages PDP_t to disassemble all parts, and if unsuccessful, it uses PDP_r to disassemble again. We denote SBDP* as DBG-guided SBDP with (f_c, f_a) and DBGs constraints. All experiments were conducted on a cloud computer with clock speeds of 2.00 GHz Xeon processors and processes time out after 2 hours. In PDP*, both translational and rotational search procedures are allotted 2 hours each. Physical simulation is restricted to 360 seconds to prevent endless execution.

Table 1 compares the number of problems solved by different disassembly planners. SBDP* surpasses all other planners in each category and solves most of the problems, 4135 out of 4196 problems, accounting for 98.55% of the total. SBDP* and SBDP consistently outperform PDP*, PDP_r and PDP_t showcasing the effectiveness of state-based search in disassembly planning. Utilizing DBGs information, SBDP* successfully addresses 36 problems unsolved by SBDP, emphasizing the significance of DBGs. Compared with SBDP*, SBDP solves extra 7 problems, and PDP* and PDP_r each solves a unique problem unsolved by SBDP*. In these problems, we empirically observed occasional incorrect collision detection, resulting in penetration disassembly trajectories for all approaches.

Table 2 compares SBDP*, SBDP, PDP*, and PDP_r in terms of search efficiency, including path planning time (PT) and disassembly planning time (DT), as well as search space, indicated by the number of executed simulations (Sim) across commonly solved problems. Due to the limited problem coverage from Table 1, PDP_t is excluded. We note that path planning time only counts the simulation cost while disassembly planning time considers the overall cost, i.e. static analysis, DBGs updates, and path planning in SBDP*. In PDP*, only successful planning time and executed simulations are recorded, excluding failures in translational or rotational search. SBDP* consistently outperforms all other planners, with the lowest average and standard deviation search space, and the lowest average planning time, achieving reductions of 55%/80%/75% in average search space, 40%/50%/70% in average path planning time, and 20%/35%/50% in average disassembly planning time compared to SBDP, PDP*, and PDP_r, across all problems. In the medium and large groups, SBDP* exhibits notable efficiency by reducing the average search space, path planning time, and disassembly planning time by 55%, 45%, and 25% compared to SBDP, and by 70%, 55%, and 40% compared to the best performance of PDP* and PDP_r. SBDP maintains the same advantages as SBDP* in terms of search space and planning time compared to the best performance of PDP* and PDP_r, reducing the average search space by 45%, path planning time by 20%, and disassembly planning time by

	Small (2550)						Medium (1339)						Large (65)						All (3954)					
	Sim		PT(s)		DT(s)		Sim		PT(s)		DT(s)		Sim		PT(s)		DT(s)		Sim		PT(s)		DT(s)	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
SBDP*	**12**	**72**	**17**	171	**68**	187	**55**	212	**83**	220	**163**	283	**177**	228	**493**	539	**703**	639	**29**	142	**47**	210	**111**	254
SBDP	26	171	24	204	71	218	125	366	149	443	219	542	373	526	729	771	883	823	65	269	78	337	134	394
PDP*	71	1K	27	212	87	248	312	2.8K	185	476	266	519	472	875	1K	1.3K	1.2K	1.4K	159	1.8K	97	394	166	435
PDP$_r$	25	151	27	**101**	88	**158**	249	789	335	695	417	740	825	2.5K	1.6K	1.8K	1.7K	1.8K	114	588	157	524	226	564

Table 2: Comparisons of SBDP*, SBDP, PDP*, and PDP$_r$ in terms of avg. and std-dev. number of simulation executions (Sim), path planning time (PT), and dissasembly planning time (DT) across **commonly** solved problems. K is 10^3. Best results are in bold.
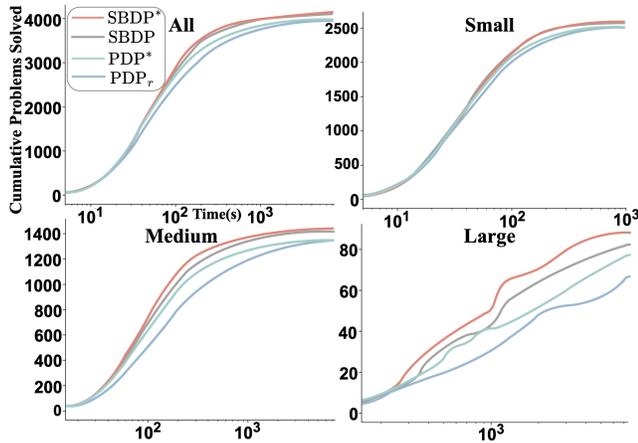

Figure 4: Coverage of solved problems as a function of time for planners SBDP*, SBDP, PDP* and PDP$_r$ across total problems and each assembly size category.

20% in total, and demonstrates the same outstanding performance in the medium and large groups.

The curves in Figure 4 display the number of problems solved as a function of time for different planners. SBDP* and SBDP solve more problems than PDP* and PDP$_r$ even for short time windows. This is especially evident in the medium and large assembly categories. This trend aligns with the results from Table 1 and Table 2, highlighting the benefits of state-based search and DBGs. SBDP* demonstrates state-of-the-art performance in terms of success rate and computational efficiency.

Discussion

By using static analysis and updates on DBGs with state information, DBG-guided state-based disassembly planning consistently outperforms the current state-of-the-art disassembly planner. SBDP benefits from state-based search and DBG-derived evaluation functions. However, applying mature planning techniques, such as classical planners with search heuristics (Hoffmann 2001; Helmert 2006; Lei and Lipovetzky 2021), in SBDP remains challenging. In SBDP, effects become available only after physical simulation, with no declarative representation available. Recent studies on leveraging *novelty*-based algorithms over Functional

STRIPS representation for solving planning problems with simulators and numerical features, such as classical control problems (Lipovetzky 2021), provide potential improvement approaches. DBGs information is non-trivial for complex disassembly problems with numerous components, as the intricate blockages among different parts constrain the disassembly sequence and path. As a result, a comprehensive analysis of precedence constraints via DBGs is necessary to determine the available disassembly strategy, which is missed in PDP. We note that our DBGs initialization concentrates solely on contacting parts and uses SDF for efficient collision detection, completing within seconds.

The state-based disassembly planning can be readily applied to assembly-by-disassembly tasks by reversing the disassembly sequence and paths and connecting them with the initial states in the assembly task to construct the entire assembly sequence and paths. For each component, a collision-free path from its initial state in the assembly task to its disassembled state in the disassembly task can be easily generated through motion planning algorithms such as RRT-Connect (Kuffner and LaValle 2000) given the existence of fewer motion constraints between these two states (Tian et al. 2022). The assembly path can be obtained through connecting the generated path with the reversed disassembly path. Subsequently, by reverse looping over the disassembly sequence and repeating the same assembly path generation method, we can ultimately derive an ordered assembly sequence containing all assembly paths.

Conclusion

We introduce the state-based search paradigm for disassembly planning (SBDP) which avoids redundant simulations of explored states. SBDP integrates both translational and rotational motions without requiring user input to choose between them. We show that new evaluation functions derived from DBGs improve the success rate and efficiency of SBDP across assemblies with varying number of components. DBG-guided SBDP demonstrates state-of-the-art performance in disassembly planning. In the future, other learning-based methods (Zakka et al. 2020; Tian et al. 2023) can be incorporated to guide SBDP. Additionally, disassembly plans generated by SBDP can be employed in assembly training systems (Zhu and Hu 2018) to facilitate robot learning through demonstrations (Thomas et al. 2018; Fan, Luo, and Tomizuka 2019; De Winter et al. 2021).

Acknowledgements

References

Aguinaga, I.; Borro, D.; and Matey, L. 2008. Parallel RRT-Based Path Planning for Selective Disassembly Planning. *The International Journal of Advanced Manufacturing Technology*, 36: 1221–1233.

Bedeoui, A.; Benhadj, R.; Trigui, M.; and Aifaoui, N. 2018. Assembly plans generation of complex machines based on the stability concept. *Procedia CIRP*, 70: 66–71.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence*, 129(1-2): 5–33.

Chen, W.-C.; Tai, P.-H.; Deng, W.-J.; and Hsieh, L.-F. 2008. A three-stage integrated approach for assembly sequence planning using neural networks. *Expert Systems with Applications*, 34(3): 1777–1786.

De Winter, J.; EI Makrini, I.; Van de Perre, G.; Nowé, A.; Verstraten, T.; and Vanderborght, B. 2021. Autonomous Assembly Planning of Demonstrated Skills with Reinforcement Learning in Simulation. *Autonomous Robots*, 45: 1097–1110.

Ebinger, T.; Kaden, S.; Thomas, S.; Andre, R.; Amato, N. M.; and Thomas, U. 2018. A General and Flexible Search Framework for Disassembly Planning. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation*, ICRA, 3548–3555.

Fan, Y.; Luo, J.; and Tomizuka, M. 2019. A Learning Framework for High Precision Industrial Assembly. In *Proceedings of the 2019 IEEE International Conference on Robotics and Automation*, ICRA, 811–817.

Ghandi, S.; and Masehian, E. 2015. Review and Taxonomies of Assembly and Disassembly Path Planning Problems and Approaches. *Computer-Aided Design*, 67: 58–86.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI Magazine*, 22: 57.

Homem de Mello, L.; and Sanderson, A. 1991. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation*, 7: 228–240.

Kuffner, J. J.; and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, ICRA, 995–1001.

LaValle, S. 1998. Rapidly-Exploring Random Trees: A New Tool for Path Planning. *Technical Report 98-11*.

Lei, C.; and Lipovetzky, N. 2021. Width-based backward search. In *Proceedings of the 31st International Conference on Automated Planning and Scheduling*, ICAPS, 219–224.

Lipovetzky, N. 2021. Planning for novelty: Width-based algorithms for common problems in control, planning and reinforcement learning. *arXiv preprint arXiv:2106.04866*.

Lozano-Perez, T.; and Wilson, R. H. 1993. Assembly Sequencing for Arbitrary Motions. In *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, ICRA, 527–532.

Perrard, C.; Lehmann, O.; Bonjour, E.; and Dalla-Zuanna, C. 2023. A new method for functional assembly plan generation and evaluation. Implementation in CapLog, an efficient software. *The International Journal of Advanced Manufacturing Technology*, 1–28.

Thomas, G.; Chien, M.; Tamar, A.; Ojea, J. A.; and Abbeel, P. 2018. Learning Robotic Assembly from CAD. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation*, ICRA, 3524–3531.

Tian, Y.; Willis, K. D. D.; Omari, B. A.; Luo, J.; Ma, P.; Li, Y.; Javid, F.; Gu, E.; Jacob, J.; Sueda, S.; Li, H.; Chitta, S.; and Matusik, W. 2023. ASAP: Automated Sequence Planning for Complex Robotic Assembly with Physical Feasibility. *arXiv preprint arXiv:2309.16909*.

Tian, Y.; Xu, J.; Li, Y.; Luo, J.; Sueda, S.; Li, H.; Willis, K. D.; and Matusik, W. 2022. Assemble Them All: Physics-Based Planning for Generalizable Assembly by Disassembly. *ACM Transactions on Graphics*, 41: 1–11.

Wang, H.; Rong, Y.; and Xiang, D. 2014. Mechanical Assembly Planning Using Ant Colony Optimization. *Computer-Aided Design*, 47: 59–71.

Wang, L.; Hou, Y.; Li, X.; and Sun, S. 2013. An enhanced harmony search algorithm for assembly sequence planning. *International Journal of Modelling, Identification and Control*, 18(1): 18–25.

Wilson, R. H. 1992. *On geometric assembly planning*. Phd thesis, Stanford University.

Wilson, R. H.; Kavraki, L.; Latombe, J.-C.; and Lozano-Pérez, T. 1995. Two-Handed Assembly Sequencing. *The International journal of robotics research*, 14: 335–350.

Xu, J.; Chen, T.; Zlokapa, L.; Foshey, M.; Matusik, W.; Sueda, S.; and Agrawal, P. 2021. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of the 17th Robotics: Science and Systems*, RSS.

Zakka, K.; Zeng, A.; Lee, J.; and Song, S. 2020. Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation*, ICRA, 9404–9410.

Zeng, C.; Gu, T.; Zhong, Y.; and Cai, G. 2011. A multi-agent evolutionary algorithm for connector-based assembly sequence planning. *Procedia Engineering*, 15: 3689–3693.

Zhu, Z.; and Hu, H. 2018. Robot Learning from Demonstration in Robotic Assembly: A Survey. *Robotics*, 7: 17.

Zickler, S.; and Veloso, M. M. 2009. Efficient Physics-Based Planning: Sampling Search Via Non-Deterministic Tactics and Skills. In *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS, 27–33.